A Fast and Effective Timing Driven Placement Tool for FPGAs

Mohammed A. S. Khalid and Yonghong Xu

Department of Electrical and Computer Engineering, University of Windsor

401 Sunset Avenue, Windsor, Ontario, CANADA N9B 3P4

{mkhalid, xu1j} @ uwindsor.ca

Abstract. In this paper, we present TQPF, a **T**iming-Driven **Q**uadratic-based **P**lacement Tool for **F**PGAs. Quadratic placement algorithms try to minimize total squared wire length by solving linear equations. The resulting placement tends to locate all cells near the center of the chip with a large amount of overlap. Also, since squared wire length is only an indirect measure of linear wire length, the resulting total wire length may not be minimized. We propose methods to alleviate the above two problems that give high quality results while minimizing the total run time. We incorporate multiple iterations of equation solving process together with a technique for pulling nodes out of the dense area while minimizing linear wire length. Experimental results using twenty Microelectronics Center of North Carolina (MCNC) benchmark circuits show that, on average, TQPF is approximately three times faster than Versatile Placement and Routing tool for FPGAs (VPR). The estimated total wire length, on average, is only 1.4% longer and the critical path delay is 4.9% lower.

1. Introduction

Placement is an important physical design step in the mapping of circuits to FPGAs. Due to improvements in fabrication technology, logic capacity of FPGAs is increasing. Consequently placement run times for

currently available tools is also increasing significantly. This motivates us to explore new algorithms and heuristics for FPGA placement that provide equal or better quality while minimizing the run time.

In recent years, many placement algorithms have been proposed to handle wire length and delay minimization problem in VLSI chips and FPGAs. These include partitioning-based placement algorithms[3][9], quadratic placement algorithms[1][4][5][6] and simulated annealing based placement algorithms[2][7][8]. Simulated annealing based placement algorithms give high quality results at the expense of relatively long run times. Quadratic placement algorithms are fast and hence capable of handling large designs. It is desirable to achieve low computational complexity while maintaining the high quality of simulated annealing based placement.

Quadratic placement algorithms use squared wire length as the objective function and try to minimize it by solving linear equations. Although quadratic wire length is only an indirect measure of the linear wire length, quadratic placement can minimize the quadratic wire length efficiently. As a result, it is widely used in VLSI placement[1][4][5][6]. The main concern with quadratic placement is that the positions of the nodes we get from the linear equation solver tends to locate in the center of the placement area with a large amount of overlap among nodes. To deal with this overlap problem, a bisection technique is used in [1] to recursively divide the circuit and add more linear constraints to pull the nodes into center of corresponding partitions. Repelling forces are added in [10] for nodes sharing a net to maintain a target distance between them and attractive forces are added by fixed dummy nodes to pull nodes from the center to the sparse regions. In [11] spreading forces are added to pull the nodes out of the dense regions. Another concern with the quadratic placement is its quality. Because the objective function of the quadratic placement is squared wire length, not linear wire length, its quality is not optimized. To alleviate this problem, [6] adds some linear aspects when

building up the quadratic model to improve its quality and [11] uses half perimeter adjustment to improve its result. But since its main concern is to speed up the equation solving process of the quadratic placement, this technique is not well discussed. All the Quadratic algorithms discussed above target standard cell VLSI placement and focus on the wirelength minimization only. They do not really consider circuit delay at the same time while minimizing the wirelength

Since the architecture of FPGAs is significantly different from standard cell VLSI, the physical design methods developed for standard cells cannot be directly applied to FPGAs. For example, for standard cell VLSI placement cell overlaps are removed in the last step. Since FPGA has fixed positions for configurable logic blocks (CLBs), in TQPF algorithm we legalize the placement in intermediate steps as well. Unlike partitioning based placement[9] and simulated annealing based placement algorithm[8], quadratic placement algorithms have not been fully investigated for FPGAs. In this paper, we propose TQPF, a fast timing-driven quadratic placement based algorithm for FPGAs. It uses a post-quadratic-placement refinement step based on linear wire length reduction. We incorporate multiple iterations of the equation solving process together with linear wire length reduction process to pull the nodes out of the dense area while maintaining the minimized linear wire length. And we also incorporate a time driven simulated annealing stage to finally refine the placement. Experimental results using 20 MCNC benchmark circuits show that TQPF can achieve 2.9x speedup in placement runtime, 4.9% reduction in critical circuit delay at the expense of only 1.4% increase in total wirelength..

The rest of this paper is organized as follows. Section 2 presents the basic ideas used in a quadratic placement algorithm and time driven simulated annealing algorithm. Section 3 gives an overview of our placement algorithm and Section 4 discusses the technique we use to pull nodes out of the dense center area while

maintaining the minimized quadratic wire length. The linear wire length reduction technique is discussed in Section 5. Experimental results are presented in Section 6 and we conclude in Section 7 with a summary and a brief discussion on future work.

2. Essentials of Quadratic Placement

2.1 Essentials of Quadratic Placement

A quadratic placer takes a hyper-graph netlist as its input and produces a placement of nodes on target chip such that the total squared wire length is minimized. Quadratic placement uses the following objective function

$$\Phi(x,y) = \frac{1}{2}\sum_{i,j} W_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2] \quad (1)$$

where x, y are the coordinates of a logic block of the netlist. W_{ij} is the weight of the edge that connects node (x_i,y_i) and node (x_j,y_j). Since the input of the quadratic placer is usually represented by a hypergraph, and two nodes can be connected by more than one net, we have to convert the hypergraph into a weighted graph first. Two models are used to convert the hypergraph into a graph. Clique model introduces an edge of weight 2/p between every pair of nodes incident to a p-pin net. While Star model created a new node in the center of its net. [11] has shown that the total squared wire length will be the same for any placement under either the star model or the clique model.

The objective function can be rewritten in matrix notation as:

$$\Phi(x,y) = \frac{1}{2}x^T Qx + d_x^T x + \frac{1}{2}y^T Qy + d_y^T y + const \quad (2)$$

where Q is an n x n symmetric matrix and d_x d_y are n-dimensional vectors.

Since the objective function can be separated into x y dimensions. Only one dimension is considered. Then we get

$$\Phi(x) = \frac{1}{2} x^T Q x + d_x^T x + const \quad (3)$$

To find the minimum value of this objective function, we perform $\nabla\Phi(x) = 0$ and get the matrix equation:

$$\mathbf{Q}x + \mathbf{d}_x = 0 \quad (4)$$

This is the quadratic equation that minimizes the total squared wire length of the placement. To obtain the matrix **Q** and vector **d**, let q_{ij} be the entry in row i and column j of matrix **Q**, and d_i be the ith element of vector **d**. For two movable nodes, the cost can be rewritten as $\frac{1}{2} W_{i,j}[x_i^2 + x_j^2 - 2x_i x_j]$. The first and second terms contribute $W_{i,j}$ to q_{ii} and q_{jj} respectively. The third term contributes $W_{i,j}$ to q_{ij} and q_{ji}. For a connection between a movable node and a fixed node the cost is $\frac{1}{2} W_{i,f}[x_i^2 + x_f^2 - 2x_i x_f]$. The first term contributes W_{if} to q_{ii}, the third term contributes to $-W_{if}x_f$ to the vector d_x at row I, and the second term becomes the constant part of expression (2) after derivative operation.

The matrix **Q** is positive definite if all movable nodes are connected to fixed nodes, i.e. I/O (Input/Output) pads, either directly or indirectly. This condition holds for all real circuits since each node of the circuit should be accessible from the outside of the circuit. So the matrix equation can be solved by using non-stationary iterative methods [12].

2.2 Essentials of Simulated Annealing Placement

Simulated annealing is a well-developed and widely used algorithm for solving combinatorial optimization problems, including those used in CAD for VLSI physical design automation. As the name suggests, this algorithm mimics the annealing process used to gradually cool molten metal to produce high quality metal

structures. An ideally annealed crystal should be in the lowest energy state, which corresponds to an optimum configuration in a combinatorial optimization problem.

A simulated annealing based placement algorithm initially places logic blocks and I/Os of the circuit randomly on the FPGA chip. A parameter temperature is used to determine the probability of whether configurations that reduce the quality of the placement will be accepted. Given a random initial placement, a source logic block is chosen randomly. Then, a target location is chosen at random for this logic block within the displacement range. If the target location is occupied, then the target logic block is swapped with the source logic block and the cost of the swap is evaluated. And if the target location is empty, the source logic block is placed in the target location and the new placement is evaluated. If the new cost is less than the cost of the previous placement, the move or swap is accepted. Otherwise, the move or swap is only accepted with a small probability. As the placement quality improves, the temperature is gradually reduced, make it less likely for bad moves and swaps that degrade the placement will be accepted. Eventually, the value of temperature will be reduced to a low value such that only the moves and swaps that improve the placement quality will be accepted, making the heuristic greedy at that point.

3. Overview of the TQPF algorithm

Our algorithm reads in a technology mapped and packed circuit [2]. It consists of three stages as illustrated below. The goal of the first stage is to obtain a good initial placement. In this stage, we repeatedly build up, modify and solve linear equations to get a progressively better placement. In each iteration, we first build up quadratic equations and solve them to get the coordinates of every node of the circuit. Then we map the nodes into the entire chip area according to their current coordinates. Finally we use these new positions as reference and add extra dummy nodes to the circuit. These new nodes are used to modify the coefficient

matrix in the next iteration. This will be discussed in detail in Section 4. During each iteration, we also perform linear adjustment to minimize the linear wire length as well as the squared wire length. This is achieved by modifying the coefficient matrix of quadratic equations, which will be discussed in details in Section 5. Stage one is performed until no significant improvement can be achieved.

TQPF Algorithm

Stage 1

- Build and solve quadratic equation
- Map the circuit to the FPGA chip
- Add dummy nodes and expand the placement
- Refinement for minimizing linear wire length
- Repeat above steps until there is no more improvement

Stage 2

- Refinement for minimizing linear wire length based on legal placement until there is no significant improvement
- Re-map the circuit to the FPGA chip

Stage 3

- Low temperature timing driven Simulated Annealing to refine the final placement

When we leave stage one, we have a reasonably good legal placement. In stage two, we try to improve this placement by using the similar linear wire length reduction technique. We do not build and solve the linear equations in this stage. Instead we move the nodes directly to reduce the total linear wire length. Since no

equation solver is needed in this stage, this process is much faster than the linear adjustment in stage one and we can perform more iterations and get a better refinement.

Finally, in stage 3 the placement is refined by low temperature time driven simulated annealing algorithm to further minimize the wire length.

4. Coarse Legal Placement

Quadratic placement produces a placement that minimizes the quadratic objective function. However, it ignores any overlap among different nodes. The node position assigned by the basic quadratic placement is represented by the coordinates obtained by solving the matrix equation. The node positions tend to converge in the center area of the chip region, as shown in Fig 1(a). In this section, we describe the method we used to pull the nodes in center to the sparse areas of the FPGA chip.

To pull the nodes apart from the center of FPGA, we need some extra force applied on these nodes. The question is how to determine the direction and magnitude of such forces. In our algorithm, we first map the nodes into the entire chip according to their coordinates so that we get a legal placement. Now each node occupies one position without any overlap, as illustrated in Fig 1(b). Of course, this will not be a good placement at first, but it gives the primary information about what position each node in the circuit is supposed to be relative to other nodes.

Every node in the circuit is pulled to its target position by an extra force as shown in Fig 1(c). The strength of this force, $\mathbf{F}_w$, is determined by all the other forces applied on this node by other nodes in the circuit. For each node in the circuit, we add a dummy node on the target position, with a weight $\mathbf{F}_w$. Then, we update the matrix equation with this extra information and solve it again and we get a new (illegal) placement as

illustrated in Fig 1(d). By performing this process, the nodes in the dense area will be pull out gradually and finally will be distributed evenly in the entire chip area.

By adding these dummy nodes, the circuit is gradually expanded to chip area while the quadratic wire length is still minimized. During this process, the relative position of the nodes in the circuit will change significantly in the first several iterations. After several iterations, the dummy nodes will gradually become the dominating forces and no more changes can be made.

5. Linear adjustment

As we mentioned in Section 1, one of the concerns of quadratic placement algorithm is its quality. Quadratic wire length is an indirect measure of linear wire length. Usually reduction in quadratic wire length leads to reduction in linear wire length. But in some cases the minimization of quadratic wire length does not mean the linear wire length is minimized. We will illustrate this scenario by an example. Fig 2 shows a circuit that consists of four nodes A, B, C and D. Nodes A, B, C are fixed, and node D is movable. The distance between A (or B) and C is L. When quadratic wire length model is applied, the optimized position of node D will be in $d = (1/3) L$. Whereas when linear wire length is applied, the position will be $d = 0$, which means node D will be placed as close to A and B as possible.

A more general case is shown in Fig 3. Node A is connected to net 1, net 2, and net3. The nodes in the nets are represented by rectangle, circle, and triangle respectively. When we use quadratic objective model, we get the relative position of the nodes, as shown in the Fig 3. Since we use half perimeter to estimate the wire length of the circuit, only the nodes on the edges of each net contribute to the total wire length. In this example, node A is only the right edge of net 2 and contributes to the total wire length only through net 2. This means that if we move node A to the edge of net 3, we can reduce the wire length of net 2 while not increasing the wire length

of the other two nets. When the node is on the edge of two or more nets, the weight of nets should also be considered. Since the half perimeter of net bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals [13], the node tends to move towards larger nets. This factor is also used to assign more weight for larger net when we build the linear equations.

Every time we perform linear adjustment, we first check the circuit to find all such movable nodes and their target positions. Then if this technique is used in stage one of our algorithm, we add extra dummy nodes at the target positions and this technique will take effect when modifying the coefficient matrix in the next iteration. When it is used in stage two, we modify the coordinates of the node directly so that the relative position of the nodes is changed, and it will take effect in the next mapping process.

6. Experimental Results

In this section, we present experimental results obtained using the TQPF algorithm on 20 MCNC benchmark circuits and compare them to the results obtained using VPR.

Table 1. Benchmark circuits

Circuit name	# Blocks	# CLBs	# Nets	Chip area
alu4	1544	1522	1536	40*40
Apex2	1919	1878	1916	44*44
Apex4	1290	1262	1271	36*36
Bigkey	2133	1707	1936	54*54
Clma	8527	8383	8445	92*92
Des	2092	1591	1847	63*63
Diffeq	1600	1497	1561	39*39
Dsip	1796	1370	1599	54*54
Elliptic	3849	3735	3735	61*61
Ex1010	4618	4598	4608	68*68
Ex5p	1135	1064	1072	33*33
Frisc	3692	3556	3576	60*60
Misex3	1425	1397	1411	38*38

Pdc	4631	4575	4591	68*68
S298	1941	1931	1935	44*44
S38417	6541	6406	6435	81*81
S38584	6789	6447	6485	81*81
Seq	1826	1750	1791	42*42
Spla	3752	3690	3706	61*61
tseng	1221	1047	1099	33*33

The TQPF algorithm was implemented in the C programming language. We incorporated TQPF into publicly available VPR code version 4.30 and used the same FPGA architecture. We used conjugated gradient method for solving linear equations. We ran VPR in Timing-driven mode(TVPR). To provide a fair comparison, we randomly placed the I/O pads and provided VPR with the same I/O pad positions. The benchmark circuits are shown in Table 1. The first column shows the circuit name. The second, third and fourth columns show, respectively, the number of blocks, the number of configurable logic blocks (CLBs) and the number of nets in the circuit. A block can be an I/O pad or a CLB. A CLB contains a 4-LUT and a D-flip-flop. The fourth column shows the FPGA size needed, in terms of CLB array size, needed to implement this circuit.

Table 2. Comparison of TQPF and TVPR based on wire length and run time

Circuit	Critical Path Delay (ns)		Ratio	Total wire length		Ratio	Time		Ratio
	TQPF	TVPR	TQPF/ TVPR	TQPF	TVPR	TQPF/ TVPR	T QPF	T VPR	TQPF/ TVPR
Alu4	101.67	94.75	1.073	202.00	201.41	1.003	24	68	2.83
Apex2	105.98	127.00	0.835	285.52	281.13	1.016	39	125	3.21
Apex4	100.24	105.70	0.948	197.51	194.13	1.017	16	47	2.94
Bigkey	82.17	83.88	0.980	317.58	318.31	0.998	48	132	2.75
Clma	199.75	238.04	0.839	1563.27	1544.2	1.012	952	2697	2.83
Des	119.45	115.35	1.035	400.01	395.66	1.011	44	124	2.82

Diffeq	62.44	70.40	0.887	180.83	180.40	1.002	29	82	2.83
Dsip	70.98	81.56	0.870	312.25	310.61	1.005	33	89	2.70
Elliptic	116.57	116.21	1.003	617.49	614.85	1.004	215	555	2.58
Ex1010	199.21	201.08	0.991	681.21	678.30	1.004	288	868	3.01
Ex5p	103.68	100.31	1.034	189.24	186.66	1.014	13	39	3.00
Frisc	134.85	136.89	0.985	654.05	627.49	1.042	205	585	2.85
Misex3	98.08	100.96	0.971	201.91	199.47	1.012	19	56	2.95
Pdc	183.36	217.89	0.842	986.73	955.37	1.033	323	975	3.02
S298	132.51	142.73	0.928	233.32	229.07	1.019	43	107	2.49
S38417	100.63	99.46	1.012	763.44	725.79	1.052	591	1540	2.61
S38584.1	183.92	193.19	0.952	856.33	852.81	1.004	607	1716	2.83
Seq	105.19	116.54	0.903	281.19	274.22	1.025	37	103	2.78
Spla	158.65	179.39	0.884	671.72	681.63	0.985	187	615	3.29
Tseng	59.09	56.73	1.042	126.95	124.94	1.016	13	40	3.08
	Average		**0.9506**			**1.014**			**2.87**

A comparison of results obtained using TQPF and TVPR are shown in Table 2. All experiments were run on a computer based on Intel Pentium 4, 2.5 GHz processor. The first column shows the circuit name. The second column shows the critical circuit delay obtained for each circuit by TQPF and TVPR. The critical path delay ratio of TQPF and TVPR is shown in the third column. Similarly, the forth column shows the wire length obtained by TQPF and TVPR. The fifth column shows the wire length ratio of TQPF and TVPR. And columns four and five show the run time (given in seconds) and the run time ratio of TQPF and TVPR. From Table 2 we can observe that, for each circuit, the wire length obtained by TQPF is very close to that obtained by TVPR. On average, across 20 circuits, the wire length obtained by TQPF is only 1.4% more than TVPR. But TQPF gives better run time and better critical path delay compared to TVPR as shown in Table 2. On

average, across 20 circuits, TQPF runs 2.87X faster than TVPR. And the critical path delay is about 4.9 less. Thus we were able to achieve our goal that motivated our work, i.e. to achieve equal or better placement quality (measured by estimated wire length and crital path dealy) while improving the placement run time. We can see that by using the quadratic technique and linear wire length adjustment heuristics, we can quickly get a reasonably good placement and then spend only a small amount of time on low temperature simulated annealing to get high quality placement. We exploit the low computational complexity of the quadratic method and the high quality results of simulated annealing method, to achieve fast, high quality placement.

7. Conclusion and Future work

In this paper, we presented TQPF, a new timing driven placement algorithm for FPGAs based on quadratic placement. Experimental results using 20 MCNC benchmark circuits show that our algorithm achieves better quality in critical circuit delay, similar quality in wirelength than that of VPR, which is a state of the art FPGA placement tool. Our algorithm gives a much faster run time compared to VPR. It would be useful to apply novel algebraic method to solve the linear equation to further reduce the run time. It would also be useful to apply timing factor when building up quadratic equations and linear movement. Future extensions to the TQPF algorithm will explore these aspects..

8. Acknowledgments

The authors would like to thank Dr. Vaughn Betz of Altera Toronto for providing VPR source code and answering questions related to VPR.

9. References

[1] J. M. Kleinhans, G. Sigl, F.M. Johannes, “Gordian: A new global optimization/ rectangle dissection method for cell placement”, Proc. of ICCAD, pp. 506-509, 1988.

[2] V. Betz and J. Rose, “VPR: A New Packing, Placement and Routing Tool for FPGAs”, Proc. of FPL’97, 1997.

[3] D. J. H Huang and A. B. Kahng, “Partitioning-based Standard-cell Global Placement with and Exact Objective”, Proc. of ACM/IEEE ISPD, 1997

[4] I. I. Mahmoud, K. Asakura, T. Nishibu and T. Ohtsuki, “Experimental Appraisal of Linear and Quadratic Objective Functions Effect on Force Directed Method for Analog Placement”, IEICE Trans. On Fundamental of Electronics, Communications and Computer Sciences 4(E77-A) 1994, pp. 710-725

[5] C. J. Alpert, T. F. Chan, D. J. H. Huang, A. B. Kahng, I. L. Markov, P. Mulet and K. Yan, “Faster Minimization of Linear Wirelength for Global Placement”, Proc. of ISPD, 1997.

[6] G. Sigl, K. Doll, F. M. Johannes, “Analytical Placement: A Linear or a Quadratic Objective Function?”, Proc. of ACM/IEEE DAC, 1991

[7] W. Swartz and C. Sechen, “Timing Driven Placement for Large Standard Cell Circuits”, Proc. of ACM/IEEE DAC, 1995

[8] A. Marquardt, V. Betz, and J. Jose, “Timing-Driven Placement for FPGAs”, Proc. of FPGA’00, 2000.

[9] P. Maidee, C. Ababei and K Bazargan, “Fast Timing-driven Partitioning-based Placement for Island Style FPGAs”, Proc. of DAC’2003.

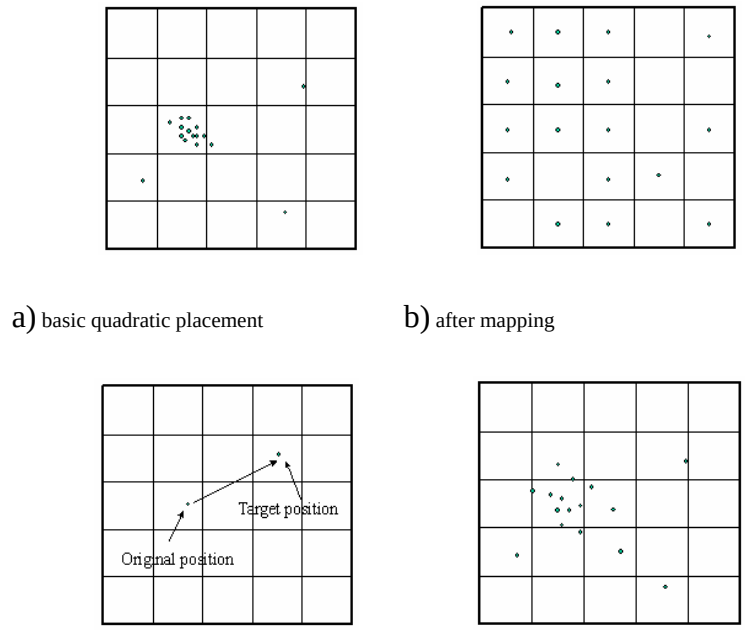
[10] H. Etawil, S. Arebi, and A. Vannelli. “Attractor-repeller approach for global placement” In Proc. IEEE/ACM ICCAD, 1999

[11] N. Viswanathan, C. Chu, “FastPlace: Efficient Analytical Placement using Cell Shifting, Iterative Local Refinement and a Hybrid Net Mode”, Proc. of ISPD, 2004

[12] R. Barret et al., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994. http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html

[13] C. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling", Proc. of ICCAD, 1994

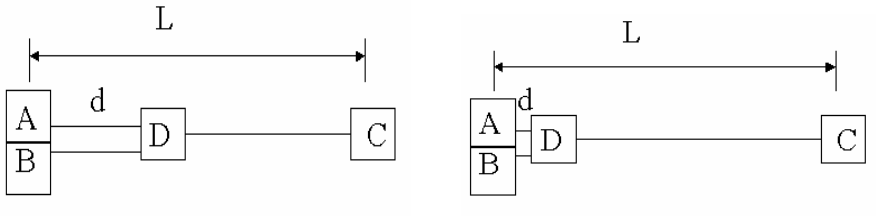
10. Figures



a) basic quadratic placement b) after mapping

c) extra force d) after extra force added

Fig. 1. Coarse legal placement process



a) Quadratic objective model b) Linear objective model

Fig. 2. Linear versus quadratic objective functions

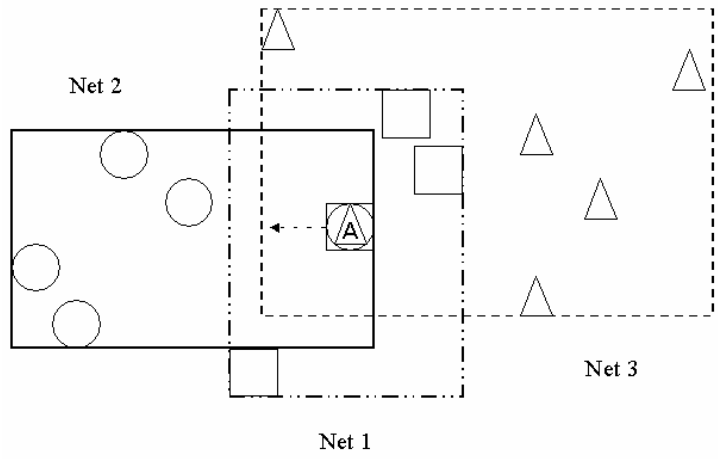


Fig. 3. Wire length contribution of one node connected to three nets